ICML 2014 AutoML Workshop

Automatic Inference for Inverting Software Simulators via Probabilistic Programming

Ardavan Saeedi ARDAVANS@MIT.EDU
Vlad Firoiu VLADFI@MIT.EDU
Vikash Mansinghka VKM@MIT.EDU

Abstract

Models of complex systems are often formalized as sequential software simulators: computationally intensive programs that iteratively build up probable system configurations given parameters and initial conditions. These simulators enable modelers to capture effects that are difficult to characterize analytically or summarize statistically. However, in many real-world applications, these simulations need to be inverted to match the observed data. This typically requires the custom design, derivation and implementation of sophisticated inversion algorithms. Here we give a framework for inverting a broad class of complex software simulators via probabilistic programming and automatic inference, using under 20 lines of probabilistic code. Our approach is based on a formulation of inversion as approximate inference in a simple sequential probabilistic model. We implement four inference strategies, including Metropolis-Hastings, a sequentialized Metropolis-Hastings scheme, and a particle Markov chain Monte Carlo scheme, requiring 4 or fewer lines of probabilistic code each. We demonstrate our framework by applying it to invert a real geological software simulator from the oil and gas industry.

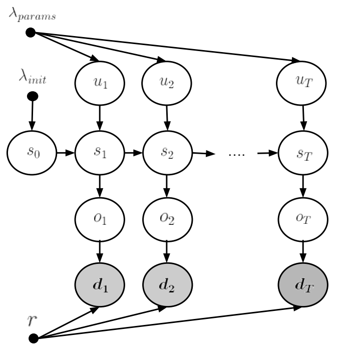
1. Introduction

Sequential software simulators are used to model complex systems in fields ranging from geophysics (Symes et al., 2011) to finance (Calvet and Fisher, 2007). They can capture dynamics that produce effects which are difficult or impossible to characterize analytically or to summarize statistically. However, the real-world problems faced by modelers often require inference, not just simulation. For example, prediction tasks require identifying realizations of a simulation that are compatible with observed data. The problem of identifying probable realizations of a simulator given data is sometimes called *simulator inversion*. Both deterministic, optimization-based methods (Boschetti et al., 1996), (Ramillien, 2001) and stochastic, sampling based (Malinverno, 2002), (Chen et al., 2006) methods are sometimes applied. Applying a standard technique to a new simulator or developing a new method for an existing simulator requires developing and implementing custom algorithms.

In this paper we show how to use probabilistic programming and automatic inference to formulate and solve a broad class of inversion problems. We define a simple interface to a sequential software simulator, and define a probabilistic model and approximate inference problem for inversion given that interface. This formulation requires under 20 lines of probabilistic code. We also describe four Monte Carlo inference strategies for solving the inversion problem, each requiring 4 or fewer lines of probabilistic code. We demonstrate our framework by applying it to invert a real geological software simulator from the oil and gas industry.

2. A framework for inverting sequential simulation software

To invert sequential simulators, we define a probabilistic model over their parameters that encodes generic priors, and use approximate inference methods to infer probable values given data.



1. s_0 = `Initialize(`λ_{init}`)` and
   $p_{init}(s_0; \lambda_{init})$ = `Prob_Init(`s_0`,`λ_{init}`)`
2. u_t = `Sample(`λ_{params}`)` and
   $p_{params}(u_t; \lambda_{params})$ = `Prob_Smp(`u_t`,`λ_{params}`)`
3. s_t = `Simulate(`s_{t-1}, u_t`)` and
   $p_{sim}(s_t; s_{t-1}, u_t)$ = `Prob_Sim(`s_{t-1}`,`u_t`)`
4. o_t = `Emit(`s_t`)` and
   $p_{emit}(o_t; s_t)$ = `Prob_Emt(`o_t`,`s_t`)`
5. $k_\gamma(o_t, r)$ = `Comp_Distance_Likelihood(`o_t, r`)`

Figure 1: **Our framework for inverting sequential simulators.** (*left*) A probabilistic graphical model that describes the inference problem corresponding to simulator inversion. Each slice corresponds to a step in the sequential simulation, capturing the dependence of the new state on the previous state and new input parameters. See main text for more details. (*right*) The procedural interface for specifying the simulator.

We assume the simulator is Markovian; that is, at every time point $t \in \{1, \ldots, T\}$, we have a state variable s_t which only depends on the previous state s_{t-1} and the parameter(s) for the state u_t: $s_t | s_{t-1}, u_t \sim p_{sim}(s_{t-1}, u_t)$ and $u_t | \lambda_{params} \sim p_{params}(\lambda_{params})$. For the initial state, we assume $s_0 | \lambda_{init} \sim p_{init}(\lambda_{init})$. Moreover, at every t given the current state s_t, an emission o_t is generated from a distribution, $o_t | s_t \sim p_{emit}(s_t)$. To afford the flexibility for many different forms of observable data, we allow simulators to come with arbitrary per-step likelihood terms. These terms are incorporated by defining a Bernoulli distribution with parameter $k_\gamma(o_t, r)$, where r is the real data and $d_t | o_t, r, \gamma \sim Br(k_\gamma(o_t, r))$. We provide an example of the $k_\gamma(o_t, r)$ in Section 2.3.

2.1. Procedural interface for specifying sequential simulators

This probabilistic model lends itself to a natural software interface that can be satisfied by many sequential simulators:

1. s_0 = `Initialize(`λ_{init}`)` and $p_{init}(s_0; \lambda_{init})$ = `Prob_Init(`s_0`,`λ_{init}`)`: These procedures return the state of the simulator at initialization and compute the probability of sampling state s_0 from the initializing distribution respectively.
2. u_t = `Sample(`λ_{params}`)` and $p_{params}(u_t; \lambda_{params})$ = `Prob_Smp(`u_t`,`λ_{params}`)`: These procedures sample the parameters for a time point u_t and compute the probability of sampling.
3. s_t = `Simulate(`s_{t-1}, u_t`)` and $p_{sim}(s_t; s_{t-1}, u_t)$ = `Prob_Sim(`s_{t-1}`,`u_t`)`: Given the current state of the simulator at time t and the parameter(s) for that time point, these procedures return the next state s_t and compute the probability of sampling.

4. o_t = `Emit(`s_t`)` and $p_{emit}(o_t; s_t)$ = `Prob_Emit(`o_t`,`s_t`)`: Given the current state at time t, these procedures emit the observation for that time point and computes the probability of emission.
5. $k_\gamma(o_t, r)$ = `Comp_Distance_Likelihood(`o_t`,` r`)` : Given the observation and the real data, this procedure calculates the probability of having an observation at time t.

2.2. A real-world example: inverting a geological forward model

We focus on a simulator developed by an oil and gas company. In this model, the states (s_t) correspond to geological features called a *lobes* and the emissions (o_t) correspond to the porosities of the substrate. The parameters u_t for each state are n-tuples of independent uniform random variables, $u_t \sim Unif[0,1]^n$. The real data r is given by a set of L *well logs*, or sequences of porosities at varying heights, each at a different location g_ℓ in the geological model. Figure 6 shows a generated sample from the geological simulator. In our dataset, well logs are available for $L = 7$ wells. The color of the lobe in renderings from the simulator represents the porosity at that lobe. See Figure 2 for a visualization of lobe formation and Figure 6 for the final output stratigraphy showing all 7 wells.

The simulator builds a lobe s_t according to a complex geological model Ψ which given s_{t-1} and u_t is deterministic. The emission at a well location, denoted by $o_{t,\ell}$, is a function Φ of the current lobe s_t and location g_ℓ. We assume the initial state s_0 is a function of a hyperparameter λ_{init} and the emissions at different locations are independent. For every emission $o_{t,\ell}$ [1], there will be a corresponding real well log for the same location and lobe $r_{t,\ell}$. We set $o_t = (o_{t,1}, \ldots, o_{t,L})$ and $r_t = (r_{t,1}, \ldots, r_{t,L})$.

The generative model for the observations at each lobe can be summarized as:

$$\begin{aligned}
s_0|\lambda_{init} &\sim p_{init}(\lambda_{init}) \\
u_t &\sim Unif[0,1]^n \\
s_t|s_{t-1}, u_t &\sim \delta_{\Psi(s_{t-1},u_t)} \\
o_{t,\ell}|s_t &\sim \delta_{\Phi(s_t,g_\ell)} \\
d_t|o_t, r, \gamma &\sim Br(k_\gamma(o_t, r))
\end{aligned}$$

The problem of stochastic inversion of a sequential simulator is finding the joint posterior of the states and the parameters given a sequence of real data r (i.e. finding $p(u_{1:t}, s_{1:t}, o_{1,t}|d_{1:t})$ in the model of Section 2). For the rest of this paper, we assume the model to be the geological model defined in this section.

2.3. Distance based likelihood function

In a complex simulator, the likelihood is intractable and an ABC filtering (Jasra et al., 2012) or ABC-MCMC (Marjoram et al., 2003) methods may be useful. However, in the geological model (without the distance based likelihood) $p_{emit}(o_t|s_t)$ and $p_{sim}(s_t|s_{t-1}, u_t)$ are delta distributions with a single atom. Thus, the likelihood is tractable. However,

1. To be more precise, every emission $o_{t,\ell}$ at well ℓ is a sequence of porosities $\nu_{h,\ell}$ indexed by height h. The height at the end of lobe t at well ℓ is denoted by $end_{t,\ell}$; hence, the generated observation at well ℓ and lobe t is given by $o_{t,\ell} = (\nu_{end_{t-1,\ell},\ell}, \nu_{end_{t-1,\ell}+1,\ell}, \ldots, \nu_{end_{t,\ell},\ell})$. Similarly, we can define the sequence of porosities for the real well logs.

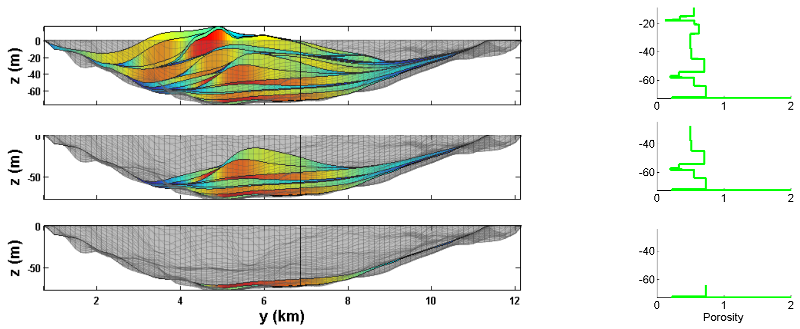


Figure 2: **Lobe formation in the model.** (*Left*) The side view of stratigraphy at a sample well location for sequential sampling of the lobes (for lobes 1, 10, and 30). (*Right*) The generated well log for the same well. The x axis is the porosity value and the y axis is the height.

the deterministic structure of the simulator can result in poor performance of any inference method. For instance, in an SMC scheme, $p_{emit}(o_t|s_t)p_{sim}(s_t|s_{t-1}, u_t)$, which appears in the weight updating equations of particles, will be zero for all the particles with probability one. This explains the reason behind defining the likelihood in terms of the distance between the real data and the generated observation.

Recall that, for lobe t and well ℓ, we denote the generated data by $o_{t,\ell}$ and the real data by $r_{t,\ell}$. We assume that the error values for different locations are independent. For each location we use the following kernel to compute the error $k_\gamma(o_{t,\ell}, r_{t,\ell}) = exp(-\gamma\|o_{t,\ell} - r_{t,\ell}\|)$ where γ is the parameter of the kernel which controls the bandwidth.

2.4. Probabilistic programming and automatic inference

We use Venture (Mansinghka et al., 2014) to represent our probabilistic model, including an interface to external simulation software. We use the automatic inference mechanisms in Venture to implement several strategies that can be applied to any simulator that satisfies the requirements of our interface. The Venture program we use for inverting all such simulators requires under 20 lines of probabilistic code.

Figure 3 shows the probabilistic code for the model and four inference strategies. We focus on inversion strategies built out of the building blocks provided by Metropolis Hastings (MH) and particle Markov chain Monte Carlo methods (PMCMC) (Andrieu et al., 2010). Each of the four strategies we present requires 4 or fewer lines of probabilistic code to implement. See (Mansinghka et al., 2014) for details regarding the syntax and semantics of Venture.

3. Experiments

We report two experimental results. First, we compare the performance of particle Gibbs, Metropolis-Hastings and sequential Metropolis-Hastings. For all three methods, we set the kernel bandwidth γ to 1. We use 10 lobes; each of these problem instances involves exploring a jagged 50-dimensional energy landscape. Figure 4 shows the results. On this problem, we find sequential Metropolis-Hastings to be the most effective, with particle Gibbs also exhibiting reasonable performance. Pure Metropolis-Hastings occasionally performs quite well but exhibits higher variance, frequently getting stuck in local minima.

```
[assume sim (make_simulator) ]
[assume get_init (lambda () (sim 'initialize)) ]
[assume get_params (mem (lambda (t) (scope_include 0 t (make_array (uniform_continuous 0 1) 5)))) ]
[assume get_state (mem (lambda (t) (sim 'simulate (get_params t) t (if (= t 0)
                                                                      (get_init)
                                                                      (get_state (- t 1))))))]
[assume get_emission (mem (lambda (t) (sim 'emit (get_state t)))))]
[assume get_distance (mem (lambda (t) (sim 'distance (get_emission t)))))]
```

```
// Particle Gibbs
for t = 1...T:
    [observe (log_flip (get_distance t)) true]
[infer (pgibbs 0 ordered 10 50)]
```

```
// Metropolis Hastings
for t = 1...T:
    [observe (log_flip (get_distance t)) true]
[infer (mh default one 500)]
```

```
// Hybrid PGibbs-MH
for t = 1...T:
    [observe (log_flip (get_distance t)) true]
[infer (cycle ((pgibbs 0 ordered 10 10)
               (mh default one 50)) 10)]
```

```
// Sequential MH
for t = 1...T:
    [observe (log_flip (get_distance t)) true]
    for i == 1...t:
        [infer (mh 0 one 10)]
```

Figure 3: **A probabilistic program implementing our framework for inverting sequential simulators.** *(Top code block)* Venture code for the probabilistic model, using a single procedure `sim` to interface with external simulation software (in our experiments , via a Python to MATLAB link). *(Middle left)* The code for loading in observations (e.g. from well logs) and for running a particle Gibbs method for 50 iterations and 10 particles. *(Middle right)* Running single-site (random scan) MH for 500 iterations. *(Bottom right)* Sequential MH with $10t$ iterations over the first t observations. In this strategy, data incorporation is interleaved with inference, to incrementally account for strong sequential dependencies. *(Bottom left)* A hybrid method based on alternating particle Gibbs and single-site (random scan) MH.

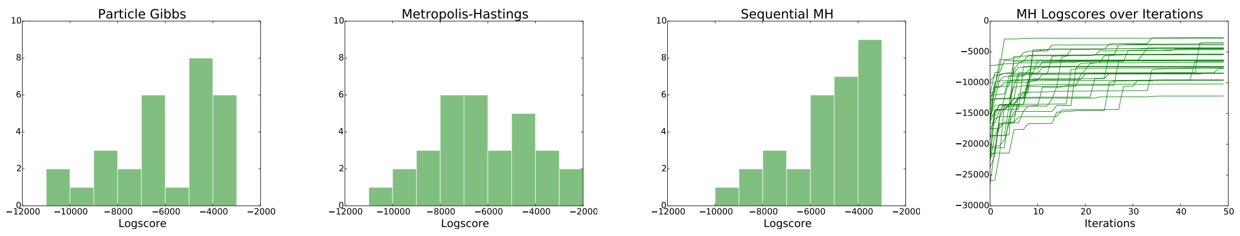


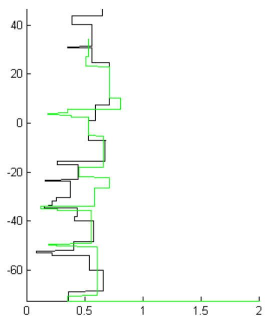
Figure 4: **Comparison of automatic inference methods on the geological simulator.** We show histograms of log probability for 30 independent runs of each method. We also show the trajectories taken by the Metropolis-Hastings method.

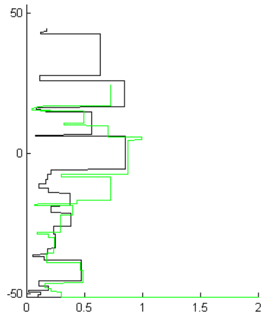
The three methods were run for roughly equivalent numbers of iterations (500), and their runtimes were all within a factor of 2 of each other. This is due to the runtime being dominated by calls to the simulator, and one iteration resulting in about 5-10 simulator calls for all three methods.

Figure 5 shows typical results for a larger-scale experiment searching over 80 lobes (400 dimensions). Many complex features of the well logs are captured by our method, although some wells are only poorly explained. Based on discussions with geologists, these results are comparable in quality to those obtained via a custom optimization-based baseline. It thus may be possible to improve accuracy as well as reduce code complexity by developing a more sophisticated inference scheme that can be automatically applied to this broad class of inversion problems.

4. Discussion

These preliminary results show that it is possible to use a general-purpose probabilistic programming system with only automatic, general-purpose inference mechanisms to invert


(a) PGibbs - Well #4


(b) PGibbs - Well #5

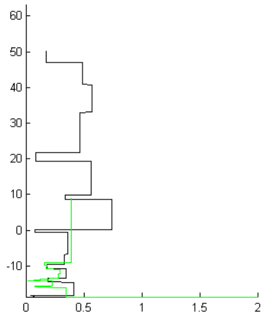

(c) PGibbs - Well #6

Figure 5: **Typical large-scale (80 lobe) well log fits**. *(a,b,c)* Inverted (green) and true (black) well logs obtained using particle Gibbs with 250 particles and 2 transitions; results correspond to the particle with highest weight.

sophisticated software simulators. A 10-line probabilistic program suffices. Multiple inference strategies can be specified with 4 or fewer lines, and can be compared to produce an ensemble of probable inversions. If the underlying simulator is changed, the only change to the probabilistic program that is needed is to generate the appropriate random parameters per simulation step. The inference programs do not need to be changed at all, even though the transition operators they induce may be quite different.

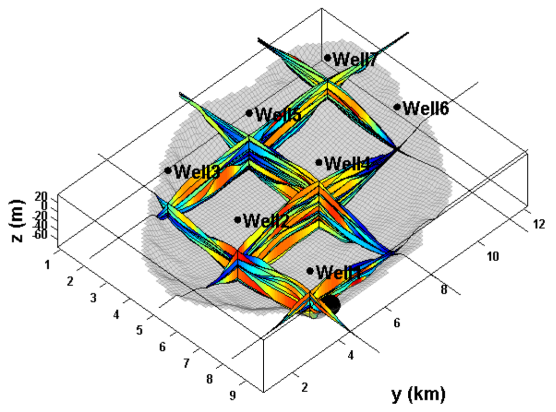


Figure 6: **Final output stratiagraphy,** showing the location of all 7 wells and many of the lobes.

Future work will investigate more sophisticated models and automatic, general-purpose inference schemes, as well as applications to other simulators. It would be especially interesting to address statistical issues in inversion, for example by augmenting our simulator interface to expose and label parameters that affect model complexity or adjust the resolution of the data, and using model selection and parameter estimation to adjust them appropriately.

Acknowledgements

We thank Shell company for the funding and provision of the lobes forward stratigraphy simulator. We also thank Zoltan Sylvester, Alessandro Cantelli, Matt Wolinsky and Oriol Falivene who built the simulator code.

References

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

Fabio Boschetti, Mike C Dentith, and Ron D List. Inversion of seismic refraction data using genetic algorithms. *Geophysics*, 61(6):1715–1727, 1996.

Laurent E Calvet and Adlai J Fisher. Multifrequency news and stock returns. *Journal of Financial Economics*, 86(1):178–212, 2007.

J Chen, Susan Hubbard, J Peterson, K Williams, M Fienen, P Jardine, and D Watson. Development of a joint hydrogeophysical inversion approach and application to a contaminated fractured aquifer. *Water Resources Research*, 42(6), 2006.

Ajay Jasra, Sumeetpal S Singh, James S Martin, and Emma McCoy. Filtering via approximate bayesian computation. *Statistics and Computing*, 22(6):1223–1237, 2012.

Alberto Malinverno. Parsimonious bayesian markov chain monte carlo inversion in a nonlinear geophysical problem. *Geophysical Journal International*, 151(3):675–688, 2002.

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *ArXiv e-prints*, March 2014.

Paul Marjoram, John Molitor, Vincent Plagnol, and Simon Tavaré. Markov chain monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26): 15324–15328, 2003.

Guillaume Ramillien. Genetic algorithms for geophysical parameter inversion from altimeter data. *Geophysical Journal International*, 147(2):393–402, 2001.

William W Symes, Dong Sun, and Marco Enriquez. From modelling to inversion: designing a well-adapted simulator. *Geophysical Prospecting*, 59(5):814–833, 2011.